1 Introduction

1.1 What Is Information Retrieval?

Information retrieval (IR) is concerned with representing, searching, and manipulating large collections of electronic text and other human-language data. IR systems and services are now widespread, with millions of people depending on them daily to facilitate business, education, and entertainment. Web search engines — Google, Bing, and others — are by far the most popular and heavily used IR services, providing access to up-to-date technical information, locating people and organizations, summarizing news and events, and simplifying comparison shopping. Digital library systems help medical and academic researchers learn about new journal articles and conference presentations related to their areas of research. Consumers turn to local search services to find retailers providing desired products and services. Within large companies, enterprise search systems act as repositories for e-mail, memos, technical reports, and other business documents, providing corporate memory by preserving these documents and enabling access to the knowledge contained within them. Desktop search systems permit users to search their personal e-mail, documents, and files.

1.1.1 Web Search

Regular users of Web search engines casually expect to receive accurate and near-instantaneous answers to questions and requests merely by entering a short query — a few words — into a text box and clicking on a search button. Underlying this simple and intuitive interface are clusters of computers, comprising thousands of machines, working cooperatively to generate a ranked list of those Web pages that are likely to satisfy the information need embodied in the query. These machines identify a set of Web pages containing the terms in the query, compute a score for each page, eliminate duplicate and redundant pages, generate summaries of the remaining pages, and finally return the summaries and links back to the user for browsing.

In order to achieve the subsecond response times expected from Web search engines, they incorporate layers of caching and replication, taking advantage of commonly occurring queries and exploiting parallel processing, allowing them to scale as the number of Web pages and users increase. In order to produce accurate results, they store a “snapshot” of the Web. This snapshot must be gathered and refreshed constantly by a *Web crawler*, also running on a cluster of hundreds or thousands of machines, and downloading periodically — perhaps once a week — a fresh copy of each page. Pages that contain rapidly changing information of high quality, such as news services, may be refreshed daily or hourly.

Consider a simple example. If you have a computer connected to the Internet nearby, pause for a minute to launch a browser and try the query "information retrieval" on one of the major commercial Web search engines. It is likely that the search engine responded in well under a second. Take some time to review the top ten results. Each result lists the URL for a Web page and usually provides a title and a short snippet of text extracted from the body of the page. Overall, the results are drawn from a variety of different Web sites and include sites associated with leading textbooks, journals, conferences, and researchers. As is common for *informational* queries such as this one, the Wikipedia article[1] may be present. Do the top ten results contain anything inappropriate? Could their order be improved? Have a look through the next ten results and decide whether any one of them could better replace one of the top ten results.

Now, consider the millions of Web pages that contain the words "information" and "retrieval". This set of pages includes many that are relevant to the subject of information retrieval but are much less general in scope than those that appear in the top ten, such as student Web pages and individual research papers. In addition, the set includes many pages that just happen to contain these two words, without having any direct relationship to the subject. From these millions of possible pages, a search engine's ranking algorithm selects the top-ranked pages based on a variety of features, including the content and structure of the pages (e.g., their titles), their relationship to other pages (e.g., the hyperlinks between them), and the content and structure of the Web as a whole. For some queries, characteristics of the user such as her geographic location or past searching behavior may also play a role. Balancing these features against each other in order to rank pages by their expected relevance to a query is an example of *relevance ranking*. The efficient implementation and evaluation of relevance ranking algorithms under a variety of contexts and requirements represents a core problem in information retrieval, and forms the central topic of this book.

1.1.2 Other Search Applications

Desktop and file system search provides another example of a widely used IR application. A desktop search engine provides search and browsing facilities for files stored on a local hard disk and possibly on disks connected over a local network. In contrast to Web search engines, these systems require greater awareness of file formats and creation times. For example, a user may wish to search only within their e-mail or may know the general time frame in which a file was created or downloaded. Since files may change rapidly, these systems must interface directly with the file system layer of the operating system and must be engineered to handle a heavy update load.

[1] `en.wikipedia.org/wiki/Information_retrieval`

Lying between the desktop and the general Web, enterprise-level IR systems provide document management and search services across businesses and other organizations. The details of these systems vary widely. Some are essentially Web search engines applied to the corporate intranet, crawling Web pages visible only within the organization and providing a search interface similar to that of a standard Web search engine. Others provide more general document- and content-management services, with facilities for explicit update, versioning, and access control. In many industries, these systems help satisfy regulatory requirements regarding the retention of e-mail and other business communications.

Digital libraries and other specialized IR systems support access to collections of high-quality material, often of a proprietary nature. This material may consist of newspaper articles, medical journals, maps, or books that cannot be placed on a generally available Web site due to copyright restrictions. Given the editorial quality and limited scope of these collections, it is often possible to take advantage of structural features — authors, titles, dates, and other publication data — to narrow search requests and improve retrieval effectiveness. In addition, digital libraries may contain electronic text generated by *optical character recognition* (OCR) systems from printed material; character recognition errors associated with the OCR output create yet another complication for the retrieval process.

1.1.3 Other IR Applications

While search is the central task within the area of information retrieval, the field covers a wide variety of interrelated problems associated with the storage, manipulation, and retrieval of human-language data:

- Document *routing*, *filtering*, and *selective dissemination* reverse the typical IR process. Whereas a typical search application evaluates incoming queries against a given document collection, a routing, filtering, or dissemination system compares newly created or discovered documents to a fixed set of queries supplied in advance by users, identifying those that match a given query closely enough to be of possible interest to the users. A news aggregator, for example, might use a routing system to separate the day's news into sections such as "business," "politics," and "lifestyle," or to send headlines of interest to particular subscribers. An e-mail system might use a spam filter to block unwanted messages. As we shall see, these two problems are essentially the same, although differing in application-specific and implementation details.
- Text *clustering* and *categorization* systems group documents according to shared properties. The difference between clustering and categorization stems from the information provided to the system. Categorization systems are provided with *training data* illustrating the various classes. Examples of "business," "politics," and "lifestyle" articles might be provided to a categorization system, which would then sort unlabeled articles into the same categories. A clustering system, in contrast, is not provided with training examples. Instead, it sorts documents into groups based on patterns it discovers itself.

- *Summarization* systems reduce documents to a few key paragraphs, sentences, or phrases describing their content. The snippets of text displayed with Web search results represent one example.
- *Information extraction* systems identify named entities, such as places and dates, and combine this information into structured records that describe relationships between these entities — for example, creating lists of books and their authors from Web data.
- *Topic detection and tracking* systems identify events in streams of news articles and similar information sources, tracking these events as they evolve.
- *Expert search* systems identify members of organizations who are experts in a specified area.
- *Question answering* systems integrate information from multiple sources to provide concise answers to specific questions. They often incorporate and extend other IR technologies, including search, summarization, and information extraction.
- *Multimedia information retrieval* systems extend relevance ranking and other IR techniques to images, video, music, and speech.

Many IR problems overlap with the fields of library and information science, as well as with other major subject areas of computer science such as natural language processing, databases, and machine learning.

Of the topics listed above, techniques for categorization and filtering have the widest applicability, and we provide an introduction to these areas. The scope of this book does not allow us to devote substantial space to the other topics. However, all of them depend upon and extend the basic technologies we cover.

1.2 Information Retrieval Systems

Most IR systems share a basic architecture and organization that is adapted to the requirements of specific applications. Most of the concepts discussed in this book are presented in the context of this architecture. In addition, like any technical field, information retrieval has its own jargon. Words are sometimes used in a narrow technical sense that differs from their ordinary English meanings. In order to avoid confusion and to provide context for the remainder of the book, we briefly outline the fundamental terminology and technology of the subject.

1.2.1 Basic IR System Architecture

Figure 1.1 illustrates the major components in an IR system. Before conducting a search, a user has an *information need*, which underlies and drives the search process. We sometimes refer to this information need as a *topic*, particularly when it is presented in written form as part

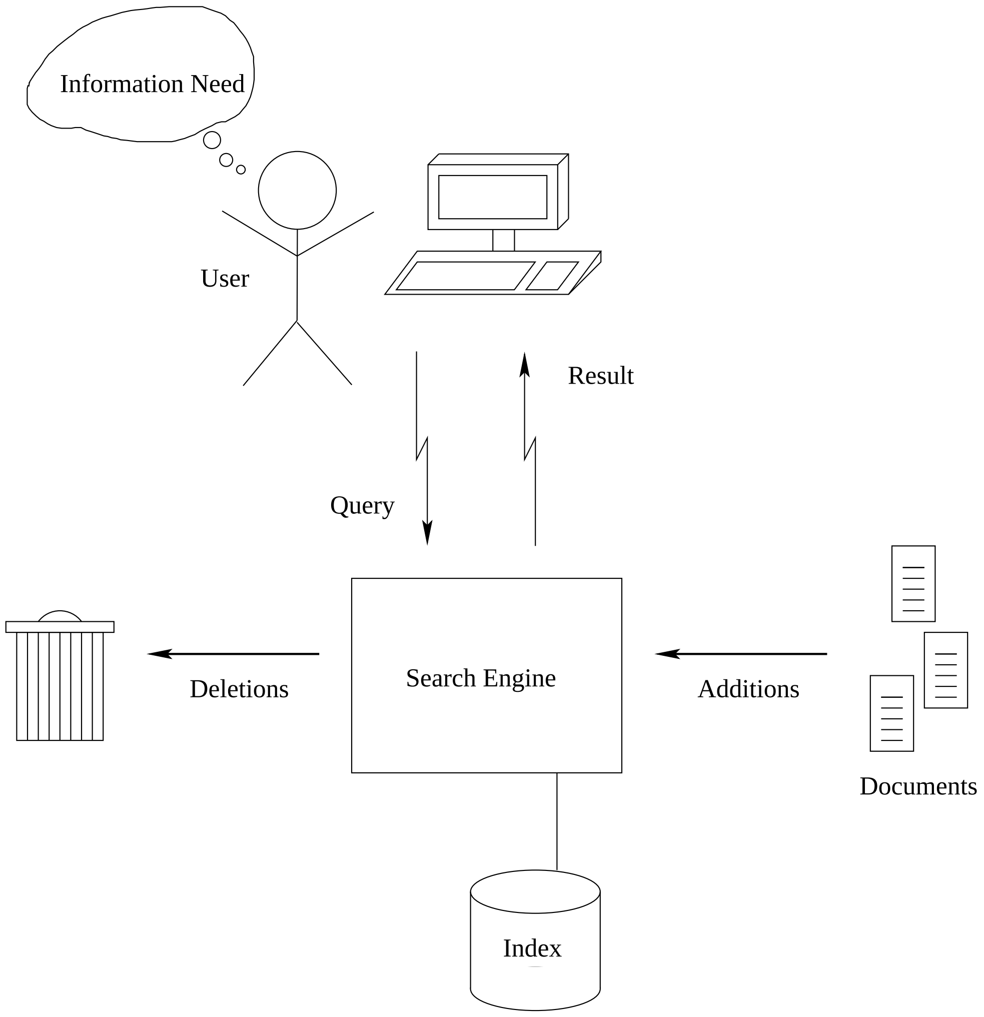


Figure 1.1 Components of an IR system.

of a test collection for IR evaluation. As a result of her information need, the user constructs and issues a *query* to the IR system. Typically, this query consists of a small number of *terms*, with two to three terms being typical for a Web search. We use "term" instead of "word", because a query term may in fact not be a word at all. Depending on the information need, a query term may be a date, a number, a musical note, or a phrase. Wildcard operators and other partial-match operators may also be permitted in query terms. For example, the term "inform*" might match any word starting with that prefix ("inform", "informs", "informal", "informant", "informative", etc.).

Although users typically issue simple keyword queries, IR systems often support a richer query syntax, frequently with complex Boolean and pattern matching operators (Chapter 5). These facilities may be used to limit a search to a particular Web site, to specify constraints on fields such as author and title, or to apply other *filters*, restricting the search to a subset of the collection. A user interface mediates between the user and the IR system, simplifying the query-creation process when these richer query facilities are required.

The user's query is processed by a *search engine*, which may be running on the user's local machine, on a large cluster of machines in a remote geographic location, or anywhere in between. A major task of a search engine is to maintain and manipulate an *inverted index* for a *document collection*. This index forms the principal data structure used by the engine for searching and relevance ranking. As its basic function, an inverted index provides a mapping between terms and the locations in the collection in which they occur. Because the size of an inverted list is on the same order of magnitude as the document collection itself, care must be taken that index access and update operations are performed efficiently.

To support relevance ranking algorithms, the search engine maintains collection statistics associated with the index, such as the number of documents containing each term and the length of each document. In addition, the search engine usually has access to the original content of the documents, in order to report meaningful results back to the user.

Using the inverted index, collection statistics, and other data, the search engine accepts queries from its users, processes these queries, and returns ranked lists of results. To perform relevance ranking, the search engine computes a *score*, sometimes called a *retrieval status value* (RSV), for each document. After sorting documents according to their scores, the result list may be subjected to further processing, such as the removal of duplicate or redundant results. For example, a Web search engine might report only one or two results from a single host or domain, eliminating the others in favor of pages from different sources. The problem of scoring documents with respect to a user's query is one of the most fundamental in the field.

1.2.2 Documents and Update

Throughout this book we use "document" as a generic term to refer to any self-contained unit that can be returned to the user as a search result. In practice, a particular document might be an e-mail message, a Web page, a news article, or even a video. When predefined components of a larger object may be returned as individual search results, such as pages or paragraphs from a book, we refer to these components as *elements*. When arbitrary text passages, video segments, or similar material may be returned from larger objects, we refer to them as *snippets*.

For most applications, the update model is relatively simple. Documents may be added or deleted in their entirety. Once a document has been added to the search engine, its contents are not modified. This update model is sufficient for most IR applications. For example, when a Web crawler discovers that a page has been modified, the update may be viewed as a deletion of the old page and an addition of the new page. Even in the context of file system search, in which individual blocks of a file may be arbitrarily modified, most word processors and other applications dealing with textual data rewrite entire files when a user saves changes. One important exception is e-mail applications, which often append new messages to the ends of mail folders. Because mail folders can be large in size and can grow quickly, it may be important to support append operations.

1.2.3 Performance Evaluation

There are two principal aspects to measuring IR system performance: *efficiency* and *effectiveness*. Efficiency may be measured in terms of time (e.g., seconds per query) and space (e.g., bytes per document). The most visible aspect of efficiency is the *response time* (also known as *latency*) experienced by a user between issuing a query and receiving the results. When many simultaneous users must be supported, the query *throughput*, measured in queries per second, becomes an important factor in system performance. For a general-purpose Web search engine, the required throughput may range well beyond tens of thousands of queries per second. Efficiency may also be considered in terms of storage space, measured by the bytes of disk and memory required to store the index and other data structures. In addition, when thousands of machines are working cooperatively to generate a search result, their power consumption and carbon footprint become important considerations.

Effectiveness is more difficult to measure than efficiency, since it depends entirely on human judgment. The key idea behind measuring effectiveness is the notion of *relevance*: A document is considered relevant to a given query if its contents (completely or partially) satisfy the information need represented by the query. To determine relevance, a human *assessor* reviews a document/topic pair and assigns a relevance value. The relevance value may be *binary* ("relevant" or "not relevant") or *graded* (e.g., "perfect", "excellent", "good", "fair", "acceptable", "not relevant", "harmful").

The fundamental goal of relevance ranking is frequently expressed in terms of the *Probability Ranking Principle* (PRP), which we phrase as follows:

> *If an IR system's response to each query is a ranking of the documents in the collection in order of decreasing probability of relevance, then the overall effectiveness of the system to its users will be maximized.*

This assumption is well established in the field of IR and forms the basis of the standard IR evaluation methodology. Nonetheless, it overlooks important aspects of relevance that must be considered in practice. In particular, the basic notion of relevance may be extended to consider the size and scope of the documents returned. The *specificity* of a document reflects the degree to which its contents are focused on the information need. A highly specific document consists primarily of material related to the information need. In a marginally specific document, most of the material is not related to the topic. The *exhaustivity* of a document reflects the degree to which it covers the information related to the need. A highly exhaustive document provides full coverage; a marginally exhaustive document may cover only limited aspects. Specificity and exhaustivity are independent dimensions. A large document may provide full coverage but contain enough extraneous material that it is only marginally specific.

When relevance is viewed in the context of a complete ranked document list, the notion of *novelty* comes to light. Once the user examines the top-ranked document and learns its relevant content, her information need may shift. If the second document contains little or no novel information, it may not be relevant with respect to this revised information need.

1.3 Working with Electronic Text

Human-language data in the form of electronic text represents the raw material of information retrieval. Building an IR system requires an understanding of both electronic text formats and the characteristics of the text they encode.

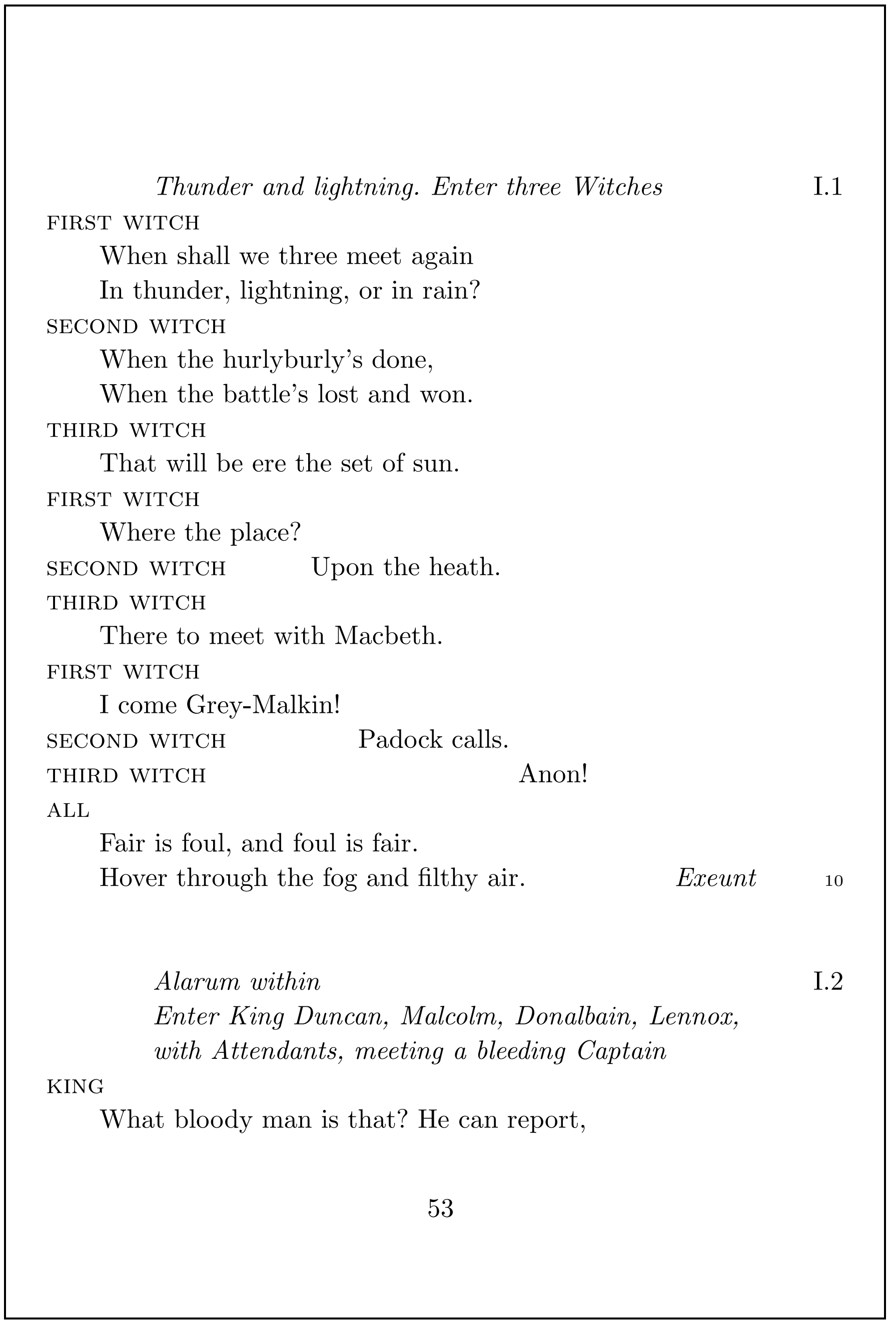
1.3.1 Text Formats

The works of William Shakespeare provide a ready example of a large body of English-language text with many electronic versions freely available on the Web. Shakespeare's canonical works include 37 plays and more than a hundred sonnets and poems. Figure 1.2 shows the start of the first act of one play, *Macbeth*.

This figure presents the play as it might appear on a printed page. From the perspective of an IR system, there are two aspects of this page that must be considered when it is represented in electronic form, and ultimately when it is indexed by the system. The first aspect, the *content* of the page, is the sequence of words in the order they might normally be read: "Thunder and lightning. Enter three Witches First Witch When shall we..." The second aspect is the *structure* of the page: the breaks between lines and pages, the labeling of speeches with speakers, the stage directions, the act and scene numbers, and even the page number.

The content and structure of electronic text may be encoded in myriad document formats supported by various word processing programs and desktop publishing systems. These formats include Microsoft Word, HTML, XML, XHTML, $\LaTeX$, MIF, RTF, PDF, PostScript, SGML, and others. In some environments, such as file system search, e-mail formats and even program source code formats would be added to this list. Although a detailed description of these formats is beyond our scope, a basic understanding of their impact on indexing and retrieval is important.

Two formats are of special interest to us. The first, HTML (HyperText Markup Language), is the fundamental format for Web pages. Of particular note is its inherent support for hyperlinks, which explicitly represent relationships between Web pages and permit these relationships to be exploited by Web search systems. Anchor text often accompanies a hyperlink, partially describing the content of the linked page.



Thunder and lightning. Enter three Witches I.1

FIRST WITCH

When shall we three meet again
In thunder, lightning, or in rain?

SECOND WITCH

When the hurlyburly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.

FIRST WITCH

Where the place?

SECOND WITCH Upon the heath.

THIRD WITCH

There to meet with Macbeth.

FIRST WITCH

I come Grey-Malkin!

SECOND WITCH Padock calls.

THIRD WITCH Anon!

ALL

Fair is foul, and foul is fair.
Hover through the fog and filthy air. *Exeunt*

Alarum within I.2
Enter King Duncan, Malcolm, Donalbain, Lennox,
with Attendants, meeting a bleeding Captain

KING

What bloody man is that? He can report,

53

Figure 1.2 The beginning of the first act of Shakespeare's *Macbeth*.

The second format, XML (eXtensible Markup Language), is not strictly a document format but rather a *metalanguage* for defining document formats. Although we leave a detailed discussion of XML to later in the book (Chapter 16), we can begin using it immediately. XML possesses the convenient attribute that it is possible to construct human-readable encodings that are reasonably self-explanatory. As a result, it serves as a format for examples throughout the remainder of the book, particularly when aspects of document structure are discussed. HTML and XML share a common ancestry in the Standard Generalized Markup Language (SGML) developed in the 1980s and resemble each other in their approach to tagging document structure.

Figure 1.3 presents an XML encoding of the start of *Macbeth* taken from a version of Shakespeare's plays. The encoding was constructed by Jon Bosak, one of the principal creators of the XML standard. Tags of the form `<`*name*`>` represent the start of a structural element and tags of the form `</`*name*`>` represent the end of a structural element. Tags may take other forms, and they may include attributes defining properties of the enclosed text, but these details are left for Chapter 16. For the bulk of the book, we stick to the simple tags illustrated by the figure.

In keeping with the traditional philosophy of XML, this encoding represents only the *logical structure* of the document — speakers, speeches, acts, scenes, and lines. Determination of the *physical structure* — fonts, bolding, layout, and page breaks — is deferred until the page is actually rendered for display, where the details of the target display medium will be known.

Unfortunately, many document formats do not make the same consistent distinction between logical and physical structure. Moreover, some formats impede our ability to determine a document's content or to return anything less than an entire document as the result of a retrieval request. Many of these formats are *binary formats*, so called because they contain internal pointers and other complex organizational structures, and cannot be treated as a stream of characters.

For example, the content of a PostScript document is encoded in a version of the programming language Forth. A PostScript document is essentially a program that is executed to render the document when it is printed or displayed. Although using a programming language to encode a document provides for maximum flexibility, it may also make the content of the document difficult to extract for indexing purposes. The PDF format, PostScript's younger sibling, does not incorporate a complete programming language but otherwise retains much of the flexibility and complexity of the older format. PostScript and PDF were originally developed by Adobe Systems, but both are now open standards and many third-party tools are available to create and manipulate documents in these formats, including open-source tools.

Various conversion utilities are available to extract content from PostScript and PDF documents, and they may be pressed into service for indexing purposes. Each of these utilities implements its own heuristics to analyze the document and guess at its content. Although they often produce excellent output for documents generated by standard tools, they may fail when faced with a document from a more unusual source. At an extreme, a utility might render a document into an internal buffer and apply a pattern-matching algorithm to recognize characters and words.

```
<STAGEDIR>Thunder and lightning. Enter three Witches</STAGEDIR>

<SPEECH>
<SPEAKER>First Witch</SPEAKER>
<LINE>When shall we three meet again</LINE>
<LINE>In thunder, lightning, or in rain?</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>Second Witch</SPEAKER>
<LINE>When the hurlyburly's done,</LINE>
<LINE>When the battle's lost and won.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>Third Witch</SPEAKER>
<LINE>That will be ere the set of sun.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>First Witch</SPEAKER>
<LINE>Where the place?</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>Second Witch</SPEAKER>
<LINE>Upon the heath.</LINE>
</SPEECH>

<SPEECH>
  <SPEAKER>Third Witch</SPEAKER>
  <LINE>There to meet with Macbeth.</LINE>
</SPEECH>
```

Figure 1.3 An XML encoding of Shakespeare's *Macbeth*.

Moreover, identifying even the simplest logical structure in PostScript or PDF poses a significant challenge. Even a document's title may have to be identified entirely from its font, size, location in the document, and other physical characteristics. In PostScript, extracting individual pages can also cause problems because the code executed to render one page may have an impact on later pages. This aspect of PostScript may limit the ability of an IR system to return a range of pages from a large document, for example, to return a single section from a long technical manual.

Other document formats are proprietary, meaning they are associated with the products of a single software manufacturer. These proprietary formats include Microsoft's "doc" format. Until recently, due to the market dominance of Microsoft Office, this format was widely used for document exchange and collaboration. Although the technical specifications for such proprietary formats are often available, they can be complex and may be modified substantially from version to version, entirely at the manufacturer's discretion. Microsoft and other manufacturers have now shifted toward XML-based formats (such as the OpenDocument format or Microsoft's OOXML), which may ameliorate the complications of indexing.

In practice, HTML may share many of the problems of binary formats. Many HTML pages include scripts in the JavaScript or Flash programming languages. These scripts may rewrite the Web page in its entirety and display arbitrary content on the screen. On pages in which the content is generated by scripts, it may be a practical impossibility for Web crawlers and search engines to extract and index meaningful content.

1.3.2 A Simple Tokenization of English Text

Regardless of a document's format, the construction of an inverted index that can be used to process search queries requires each document to be converted into a sequence of *tokens*. For English-language documents, a token usually corresponds to a sequence of alphanumeric characters (A to Z and 0 to 9), but it may also encode structural information, such as XML tags, or other characteristics of the text. Tokenization is a critical step in the indexing process because it effectively limits the class of queries that may be processed by the system.

As a preliminary step before tokenization, documents in binary formats must be converted to *raw text* — a stream of characters. The process of converting a document to raw text generally discards font information and other lower-level physical formatting but may retain higher-level logical formatting, perhaps by re-inserting appropriate tags into the raw text. This higher-level formatting might include titles, paragraph boundaries, and similar elements. This preliminary step is not required for documents in XML or HTML because these already contain the tokens in the order required for indexing, thus simplifying the processing cost substantially. Essentially, these formats are in situ raw text.

For English-language documents, characters in the raw text may be encoded as seven-bit ASCII values. However, ASCII is not sufficient for documents in other languages. For these languages, other coding schemes must be used, and it may not be possible to encode each character as a single byte. The UTF-8 representation of Unicode provides one popular method for encoding these characters (see Section 3.2). UTF-8 provides a one-to-four byte encoding for the characters in most living languages, as well as for those in many extinct languages, such as Phoenician and Sumerian cuneiform. UTF-8 is backwards compatible with ASCII, so that ASCII text is automatically UTF-8 text.

To tokenize the XML in Figure 1.3, we treat each XML tag and each sequence of consecutive alphanumeric characters as a token. We convert uppercase letters outside tags to lowercase in order to simplify the matching process, meaning that "FIRST", "first" and "First" are treated

...

745396	745397	745398	745399	745400
`<STAGEDIR>`	`thunder`	`and`	`lightning`	`enter`
745401	745402	745403	745404	745405
`three`	`witches`	`</STAGEDIR>`	`<SPEECH>`	`<SPEAKER>`
745406	745407	745408	745409	745410
`first`	`witch`	`</SPEAKER>`	`<LINE>`	`when`
745411	745412	745413	745414	745415
`shall`	`we`	`three`	`meet`	`again`
745416	745417	745418	745419	745420
`</LINE>`	`<LINE>`	`in`	`thunder`	`lightning`
745421	745422	745423	745424	745425
`or`	`in`	`rain`	`</LINE>`	`</SPEECH>`
745426	745427	745428	745429	745430
`<SPEECH>`	`<SPEAKER>`	`second`	`witch`	`</SPEAKER>`

...

Figure 1.4 A tokenization of Shakespeare's *Macbeth*.

as equivalent. The result of our tokenization is shown in Figure 1.4. Each token is accompanied by an integer that indicates its position in a collection of Shakespeare's 37 plays, starting with position 1 at the beginning of *Antony and Cleopatra* and finishing with position 1,271,504 at the end of *The Winter's Tale*. This simple approach to tokenization is sufficient for our purposes through the remainder of Chapters 1 and 2, and it is assumed where necessary. We will reexamine tokenization for English and other languages in Chapter 3.

The set of distinct tokens, or *symbols*, in a text collection is called the *vocabulary*, denoted as $\mathcal{V}$. Our collection of Shakespeare's plays has $|\mathcal{V}| = 22{,}987$ symbols in its vocabulary.

$$\mathcal{V} \;=\; \{\texttt{a}, \texttt{aaron}, \texttt{abaissiez}, ..., \texttt{zounds}, \texttt{zwaggered}, ..., \texttt{<PLAY>}, ..., \texttt{<SPEAKER>}, ..., \texttt{</PLAY>}, ...\}$$

Table 1.1 The twenty most frequent terms in Bosak's XML version of Shakespeare.

Rank	**Frequency**	**Token**	**Rank**	**Frequency**	**Token**
1	107,833	`<LINE>`	11	17,523	`of`
2	107,833	`</LINE>`	12	14,914	`a`
3	31,081	`<SPEAKER>`	13	14,088	`you`
4	31,081	`</SPEAKER>`	14	12,287	`my`
5	31,028	`<SPEECH>`	15	11,192	`that`
6	31,028	`</SPEECH>`	16	11,106	`in`
7	28,317	`the`	17	9,344	`is`
8	26,022	`and`	18	8,506	`not`
9	22,639	`i`	19	7,799	`it`
10	19,898	`to`	20	7,753	`me`

For Shakespeare, the vocabulary includes 22,943 words and 44 tags, in which we consider any string of alphanumeric characters to be a word. In this book, we usually refer to symbols in the vocabulary as "terms" because they form the basis for matching against the terms in a query. In addition, we often refer to a token as an "occurrence" of a term. Although this usage helps reinforce the link between the tokens in a document and the terms in a query, it may obscure the crucial difference between a symbol and a token. A symbol is an abstraction; a token is an instance of that abstraction. In philosophy, this difference is called the "type-token distinction." In object-oriented programming, it is the difference between a class and an object.

1.3.3 Term Distributions

Table 1.1 lists the twenty most frequent terms in the XML collection of Shakespeare's plays. Of these terms, the top six are tags for lines, speakers, and speeches. As is normally true for English text, "the" is the most frequent word, followed by various pronouns, prepositions, and other function words. More than one-third (8,336) of the terms, such as "abaissiez" and "zwaggered", appear only once.

The frequency of the tags is determined by the structural constraints of the collection. Each start tag `<`*name*`>` has a corresponding end tag `</`*name*`>`. Each play has exactly one title. Each speech has at least one speaker, but a few speeches have more than one speaker, when a group of characters speak in unison. On average, a new line starts every eight or nine words.

While the type and relative frequency of tags will be different in other collections, the relative frequency of words in English text usually follows a consistent pattern. Figure 1.5 plots the frequency of the terms in Shakespeare in rank order, with tags omitted. Logarithmic scales are used for both the x and the y axes. The points fall roughly along a line with slope -1, although both the most frequent and the least frequent terms fall below the line. On this plot, the point

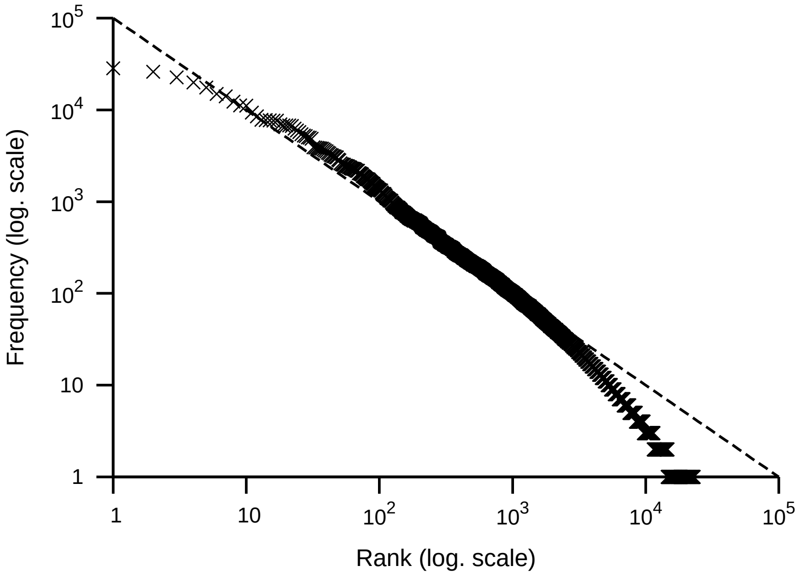


Figure 1.5 Frequency of words in the Shakespeare collection, by rank order. The dashed line corresponds to Zipf's law with $\alpha = 1$.

corresponding to "the" appears in the upper left. The point corresponding to "zwaggered" appears in the lower right, together with the other words that appear exactly once.

The relationship between frequency and rank represented by this line is known as *Zipf's law*, after the linguist George Zipf, who developed it in the 1930s and used it to model the relative frequencies of data across a number of areas in the social sciences (Zipf, 1949). Mathematically, the relationship may be expressed as

$$\log(\text{frequency}) \;=\; C - \alpha \cdot \log(\text{rank})\,, \tag{1.1}$$

or equivalently as

$$\mathcal{F}_i \sim \frac{1}{i^\alpha}\,, \tag{1.2}$$

where $\mathcal{F}_i$ is the frequency of the ith most frequent term. For English text, the value of α may vary, but it is usually close to 1. Zipf's law is known to apply to the relative word frequencies in other natural languages, as well as to other types of data. In Chapter 4, it motivates the use of a certain data structure for indexing natural language text. In Chapter 15, we apply Zipf's law to model the relative frequency of search engine queries.

1.3.4 Language Modeling

There are 912,052 tokens in Bosak's XML version of Shakespeare's plays, excluding tags (but including some front matter that does not appear in the original versions). If we pick a token uniformly at random from these plays, the probability of picking "the" is $28,317/912,052 \approx 3.1\%$, whereas the probability of picking "zwaggered" is only $1/912,052 \approx 0.00011\%$. Now, imagine that a previously unknown Shakespearean play is discovered. Can we predict anything about the content of this play from what we know of the existing plays? In making these predictions, we redefine the vocabulary $\mathcal{V}$ to exclude tags. An unknown Shakespearean play is unlikely to be encoded in XML (at least when it is first discovered).

Predictions concerning the content of unseen text may be made by way of a special kind of probability distribution known as a *language model*. The simplest language model is a fixed probability distribution $\mathcal{M}(\sigma)$ over the symbols in the vocabulary:

$$\sum_{\sigma \in \mathcal{V}} \mathcal{M}(\sigma) \;=\; 1. \tag{1.3}$$

A language model is often based on an existing text. For example, we might define

$$\mathcal{M}(\sigma) \;=\; \frac{\text{frequency}(\sigma)}{\sum_{\sigma' \in \mathcal{V}} \text{frequency}(\sigma')} \tag{1.4}$$

where frequency(σ) represents the number of occurrences of the term σ in Shakespeare's plays. Thus, we have $\mathcal{M}(\text{“the”}) \approx 3.1\%$ and $\mathcal{M}(\text{“zwaggered”}) \approx 0.00011\%$.

If we pick a token uniformly at random from Shakespeare's known plays, the probably of picking the term σ is $\mathcal{M}(\sigma)$. Based on this knowledge of Shakespeare, if we pick a token uniformly at random from the previously unseen play, we might assume that the probably of picking σ is also $\mathcal{M}(\sigma)$. If we start reading the unseen play, we can use the language model to make a prediction about the next term in the text, with $\mathcal{M}(\sigma)$ giving the probability that the next term is σ. For this simple language model, we consider each term in isolation. The language model makes the same independent prediction concerning the first term, and the next, and the next, and so on. Based on this language model, the probability that the next six terms are "to be or not to be" is:

$$2.18\% \times 0.76\% \times 0.27\% \times 0.93\% \times 2.18\% \times 0.76\% \;=\; 0.000000000069\%.$$

Equation 1.4 is called the *maximum likelihood estimate* (MLE) for this simple type of language model. In general, maximum likelihood is the standard method for estimating unknown parameters of a probability distribution, given a set of data. Here, we have a parameter corresponding to each term — the probability that the term appears next in the unseen text. Roughly speaking, the maximum likelihood estimation method chooses values for the parameters that make the data set most likely. In this case, we are treating the plays of Shakespeare as providing the necessary data set. Equation 1.4 is the assignment of probabilities that is most likely to produce

Shakespeare. If we assume that the unseen text is similar to the existing text, the maximum likelihood model provides a good starting point for predicting its content.

Language models can be used to quantify how close a new text fragment is to an existing corpus. Suppose we have a language model representing the works of Shakespeare and another representing the works of the English playwright John Webster. A new, previously unknown, play is found; experts debate about who might be its author. Assuming that our two language models capture salient characteristics of the two writers, such as their preferred vocabulary, we may apply the language models to compute the probability of the new text according to each. The language model that assigns the higher probability to the new text may indicate its author.

However, a language model need not use maximum likelihood estimates. Any probability distribution over a vocabulary of terms may be treated as a language model. For example, consider the probability distribution

$$\mathcal{M}(\text{“to”}) = 0.40 \quad \mathcal{M}(\text{“be”}) = 0.30 \quad \mathcal{M}(\text{“or”}) = 0.20 \quad \mathcal{M}(\text{“not”}) = 0.10.$$

Based on this distribution, the probability that the next six words are “to be or not to be” is

$$0.40 \times 0.30 \times 0.20 \times 0.10 \times 0.40 \times 0.30 = 0.029\%.$$

Of course, based on this model, the probability that the next six words are “the lady doth protest too much” is zero.

In practice, unseen text might include unseen terms. To accommodate these unseen terms, the vocabulary might be extended by adding an UNKNOWN symbol to represent these “out-of-vocabulary” terms.

$$\mathcal{V}' \;=\; \mathcal{V} \,\cup\, \{\text{UNKNOWN}\} \tag{1.5}$$

The corresponding extended language model $\mathcal{M}'(\sigma)$ would then assign a positive probability to this UNKNOWN term

$$\mathcal{M}'(\text{UNKNOWN}) = \beta, \tag{1.6}$$

where $0 \leq \beta \leq 1$. The value β represents the probability that the next term does not appear in the existing collection from which the model $\mathcal{M}$ was estimated. For other terms, we might then define

$$\mathcal{M}'(\sigma) \;=\; \mathcal{M}(\sigma) \cdot (1 - \beta)\,, \tag{1.7}$$

where $\mathcal{M}(\sigma)$ is the maximum likelihood language model. The choice of a value for β might be based on characteristics of the existing text. For example, we might guess that β should be roughly half of the probability of a unique term in the existing text:

$$\beta \;=\; 0.5 \cdot \frac{1}{\sum_{\sigma' \in \mathcal{V}} \text{frequency}(\sigma')}\,. \tag{1.8}$$

Fortunately, out-of-vocabulary terms are not usually a problem in IR systems because the complete vocabulary of the collection may be determined during the indexing process.

Index and text compression (Chapter 6) represents another important area for the application of language models. When used in compression algorithms, the terms in the vocabulary are usually individual characters or bits rather than entire words. Language modeling approaches that were invented in the context of compression are usually called *compression models*, and this terminology is common in the literature. However, compression models are just specialized language models. In Chapter 10, compression models are applied to the problem of detecting e-mail spam and other filtering problems.

Language models may be used to generate text, as well as to predict unseen text. For example, we may produce "random Shakespeare" by randomly generating a sequence of terms based on the probability distribution $\mathcal{M}(\sigma)$:

> *strong die hat circumstance in one eyes odious love to our the wrong wailful would all sir you to babies a in in of er immediate slew let on see worthy all timon nourish both my how antonio silius my live words our my ford scape*

Higher-order models

The random text shown above does not read much like Shakespeare, or even like English text written by lesser authors. Because each term is generated in isolation, the probability of generating the word "the" immediately after generating "our" remains at 3.1%. In real English text, the possessive adjective "our" is almost always followed by a common noun. Even though the phrase "our the" consists of two frequently occurring words, it rarely occurs in English, and never in Shakespeare.

Higher-order language models allow us to take this context into account. A *first-order* language model consists of conditional probabilities that depend on the previous symbol. For example:

$$\mathcal{M}_1(\sigma_2 \,|\, \sigma_1) \;=\; \frac{\text{frequency}(\sigma_1\sigma_2)}{\sum_{\sigma' \in \mathcal{V}} \text{frequency}(\sigma_1\sigma')} \,. \tag{1.9}$$

A first-order language model for terms is equivalent to the zero-order model for term *bigrams* estimated using the same technique (e.g., MLE):

$$\mathcal{M}_1(\sigma_2 \,|\, \sigma_1) \;=\; \frac{\mathcal{M}_0(\sigma_1\sigma_2)}{\sum_{\sigma' \in \mathcal{V}} \mathcal{M}_0(\sigma_1\sigma')} \,. \tag{1.10}$$

More generally, every nth-order language model may be expressed in terms of a zero-order $(n+1)$-gram model:

$$\mathcal{M}_n(\sigma_{n+1} \,|\, \sigma_1 \ldots \sigma_n) \;=\; \frac{\mathcal{M}_0(\sigma_1 \ldots \sigma_{n+1})}{\sum_{\sigma' \in \mathcal{V}} \mathcal{M}_0(\sigma_1 \ldots \sigma_n \, \sigma')} \,. \tag{1.11}$$

As an example, consider the phrase "first witch" in Shakespeare's plays. This phrase appears a total of 23 times, whereas the term "first" appears 1,349 times. The maximum likelihood bigram model thus assigns the following probability:

$$\mathcal{M}_0(\text{“first witch”}) \;=\; \frac{23}{912{,}051} \;\approx\; 0.0025\%$$

(note that the denominator is 912,051, not 912,052, because the total number of bigrams is one less than the total number of tokens). The corresponding probability in the first-order model is

$$\mathcal{M}_1(\text{“witch”} \,|\, \text{“first”}) \;=\; \frac{23}{1349} \;\approx\; 1.7\%.$$

Using Equations 1.9 and 1.10, and ignoring the difference between the number of tokens and the number of bigrams, the maximum likelihood estimate for “our the” is

$$\mathcal{M}_0(\text{“our the”}) \;=\; \mathcal{M}_0(\text{“our”}) \cdot \mathcal{M}_1(\text{“the”} \,|\, \text{“our”}) \;=\; 0\%,$$

which is what we would expect, because the phrase never appears in the text. Unfortunately, the model also assigns a zero probability to more reasonable bigrams that do not appear in Shakespeare, such as “fourth witch”. Because “fourth” appears 55 times and “witch” appears 92 times, we can easily imagine that an unknown play might contain this bigram. Moreover, we should perhaps assign a small positive probability to bigrams such as “our the” to accommodate unusual usage, including archaic spellings and accented speech. For example, *The Merry Wives of Windsor* contains the apparently meaningless bigram “a the” in a speech by the French physician Doctor Caius: “If dere be one or two, I shall make-a the turd.” Once more context is seen, the meaning becomes clearer.

Smoothing

One solution to this problem is to *smooth* the first-order model $\mathcal{M}_1$ with the corresponding zero-order model $\mathcal{M}_0$. Our smoothed model $\mathcal{M}_1'$ is then a linear combination of $\mathcal{M}_0$ and $\mathcal{M}_1$:

$$\mathcal{M}_1'(\sigma_2 \,|\, \sigma_1) \;=\; \gamma \cdot \mathcal{M}_1(\sigma_2 \,|\, \sigma_1) + (1-\gamma) \cdot \mathcal{M}_0(\sigma_2) \tag{1.12}$$

and equivalently

$$\mathcal{M}_0'(\sigma_1\sigma_2) \;=\; \gamma \cdot \mathcal{M}_0(\sigma_1\sigma_2) + (1-\gamma) \cdot \mathcal{M}_0(\sigma_1) \cdot \mathcal{M}_0(\sigma_2), \tag{1.13}$$

where γ in both cases is a smoothing parameter ($0 \leq \gamma \leq 1$). For example, using maximum likelihood estimates and setting $\gamma = 0.5$, we have

$$\begin{aligned} \mathcal{M}_1'(\text{“first witch”}) \;&=\; \gamma \cdot \mathcal{M}_1(\text{“first witch”}) + (1-\gamma) \cdot \mathcal{M}_0(\text{“first”}) \cdot \mathcal{M}_0(\text{“witch”}) \\ &=\; 0.5 \cdot \frac{23}{912,051} + 0.5 \cdot \frac{1,349}{912,052} \cdot \frac{92}{912,052} \;\approx\; 0.0013\% \end{aligned}$$

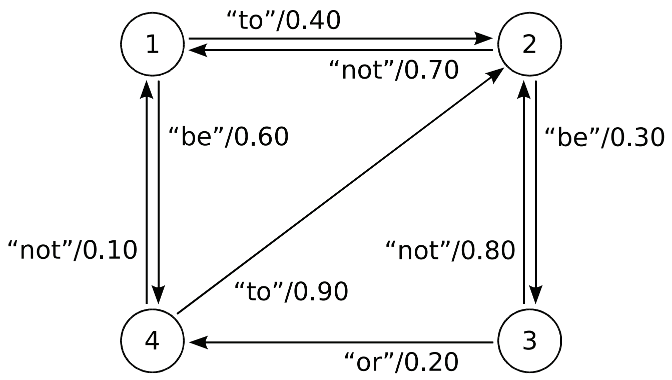


Figure 1.6 A Markov model.

and

$$\begin{aligned}\mathcal{M}'_1(\text{“fourth witch”}) &= \gamma \cdot \mathcal{M}_1(\text{“fourth witch”}) + (1-\gamma) \cdot \mathcal{M}_0(\text{“fourth”}) \cdot \mathcal{M}_0(\text{“witch”}) \\ &= 0.5 \cdot \frac{0}{912{,}051} + 0.5 \cdot \frac{55}{912{,}052} \cdot \frac{92}{912{,}052} \;\approx\; 0.00000030\%.\end{aligned}$$

First-order models can be smoothed using zero-order models; second-order models can be smoothed using first-order models; and so forth. Obviously, for zero-order models, this approach does not work. However, we can follow the same approach that we used to address out-of-vocabulary terms (Equation 1.5). Alternatively (and more commonly), the zero-order model $\mathcal{M}_{S,0}$ for a small collection S can be smoothed using another zero-order model, built from a larger collection L:

$$\mathcal{M}'_{S,0} \;=\; \gamma \cdot \mathcal{M}_{S,0} + (1-\gamma) \cdot \mathcal{M}_{L,0}\,, \tag{1.14}$$

where L could be an arbitrary (but large) corpus of English text.

Markov models

Figure 1.6 illustrates a *Markov model*, another important method for representing term distributions. Markov models are essentially finite-state automata augmented with transition probabilities. When used to express a language model, each transition is labeled with a term, in addition to the probability. Following a transition corresponds to predicting or generating that term. Starting in state 1, we may generate the string “to be or not to be” by following the state sequence $1 \to 2 \to 3 \to 4 \to 1 \to 2 \to 3$, with the associated probability

$$0.40 \times 0.30 \times 0.20 \times 0.10 \times 0.40 \times 0.30 \;=\; 0.029\%.$$

Missing transitions (e.g., between state 1 and state 4) are equivalent to transitions with zero probability. Because these transitions will never be taken, it is pointless to associate a term with them. For simplicity, we also assume that there is at most one transition between any two states. We do not sacrifice anything by making this simplification. For any Markov model with multiple transitions between the same pair of states, there is a corresponding model without them (see Exercise 1.7).

The probability predicted by a Markov model depends on the starting state. Starting in state 3, the probability of generating "to be or not to be" is

$$0.90 \times 0.30 \times 0.20 \times 0.10 \times 0.40 \times 0.30 \;=\; 0.065\%.$$

Markov models form the basis for a text compression model introduced for filtering in Chapter 10.

We may represent a Markov model with n states as an $n \times n$ *transition matrix* M, where $M[i][j]$ gives the probability of a transition from state i to state j. The transition matrix corresponding to the Markov model in Figure 1.6 is

$$M \;=\; \begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix}. \tag{1.15}$$

Note that all of the values in the matrix fall in the range $[0, 1]$ and that each row of M sums to 1. An $n \times n$ matrix with these properties is known as a *stochastic matrix*.

Given a transition matrix, we can compute the outcome of a transition by multiplying the transition matrix by a *state vector* representing the current state. For example, we can represent an initial start state of 1 by the vector (1 0 0 0). After one step, we have

$$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix} \;=\; \begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \end{pmatrix}.$$

That is, after one step, the probability of being in state 2 is 0.40 and the probability of being in state 4 is 0.60, as expected. Multiplying again, we have

$$\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \end{pmatrix} \begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix} \;=\; \begin{pmatrix} 0.34 & 0.54 & 0.12 & 0.00 \end{pmatrix}.$$

This result tells us, for example, that the probability of being in state 2 after two steps is 0.54. In general, the state vector may be any n-dimensional vector whose elements sum to 1. Multiplying the state vector by the transition matrix k times gives the probability of being in each state after k steps.

A stochastic matrix together with an initial state vector is known as a *Markov chain*. Markov chains are used in the presentation of Web link analysis algorithms in Chapter 15. Markov chains — and by extension Markov models — are named after the Russian statistician Andrey Markov (1856–1922), who stated and proved many of their properties.

1.4 Test Collections

Although *Macbeth* and Shakespeare's other plays provide an excellent source of examples for simple IR concepts, researchers have developed more substantial test collections for evaluation purposes. Many of these collections have been created as part of TREC[2] (Text REtrieval Conference), a series of experimental evaluation efforts conducted annually since 1991 by the U.S. National Institute of Standards and Technology (NIST). TREC provides a forum for researchers to test their IR systems on a broad range of problems. For example, more than 100 groups from universities, industry, and government participated in TREC 2007.

In a typical year, TREC experiments are structured into six or seven *tracks*, each devoted to a different area of information retrieval. In recent years, TREC has included tracks devoted to enterprise search, genomic information retrieval, legal discovery, e-mail spam filtering, and blog search. Each track is divided into several tasks that test different aspects of that area. For example, at TREC 2007, the enterprise search track included an e-mail discussion search task and a task to identify experts on given topics. A track typically operates for three or more years before being retired.

TREC provides at least two important benefits to the IR community. First, it focuses researchers on common problems, using common data, thus providing a forum for them to present and discuss their work and facilitating direct inter-system comparisons. What works and what does not work can be determined quickly. As a result, considerable progress and substantial performance improvements are often seen immediately following the introduction of a new track into TREC. As a second benefit, TREC aims to create *reusable* test collections that can be used by participating groups to validate further improvements and by non-participating groups to evaluate their own work. In addition, since its inception, TREC has formed the inspiration for a number of similar experimental efforts throughout the world. These include the European INEX effort for XML retrieval, the CLEF effort for multi-lingual information retrieval, the Japanese NTCIR effort for Asian language information retrieval, and the Indian FIRE effort.

[2] `trec.nist.gov`

```
<DOC>
<DOCNO> LA051990-0141 </DOCNO>

<HEADLINE> COUNCIL VOTES TO EDUCATE DOG OWNERS </HEADLINE>

<P>
The City Council stepped carefully around enforcement of a dog-curbing
ordinance this week, vetoing the use of police to enforce the law.
</P>

...

</DOC>
```

Figure 1.7 Example TREC document (LA051990-0141) from disk 5 of the TREC CDs.

1.4.1 TREC Tasks

Basic search tasks — in which systems return a ranked list from a static set of documents using previously unseen topics — are referred to as "adhoc" tasks in TREC jargon (often written as one word). Along with a set of documents, a test collection for an adhoc task includes sets of topics, from which queries may be created, and sets of relevance judgments (known as "qrel files" or just "qrels"), indicating documents that are relevant or not relevant to each topic. Over the history of TREC, adhoc tasks have been a part of tracks with a number of different research themes, such as Web retrieval or genomic IR. Despite differences in themes, the organization and operation of an adhoc task is basically the same across the tracks and is essentially unchanged since the earliest days of TREC.

Document sets for older TREC adhoc tasks (before 2000) were often taken from a set of 1.6 million documents distributed to TREC participants on five CDs. These disks contain selections of newspaper and newswire articles from publications such as the *Wall Street Journal* and the *LA Times*, and documents published by the U.S. federal government, such as the *Federal Register* and the *Congressional Record*. Most of these documents are written and edited by professionals reporting factual information or describing events.

Figure 1.7 shows a short excerpt from a document on disk 5 of the TREC CDs. The document appeared as a news article in the *LA Times* on May 19, 1990. For the purposes of TREC experiments, it is marked up in the style of XML. Although the details of the tagging schemes vary across the TREC collections, all TREC documents adhere to the same tagging convention for identifying document boundaries and document identifiers. Every TREC document is surrounded by `<DOC>`...`</DOC>` tags; `<DOCNO>`...`</DOCNO>` tags indicate its unique identifier. This identifier is used in qrels files when recording judgments for the document. This convention simplifies the indexing process and allows collections to be combined easily. Many research IR systems provide out-of-the-box facilities for working with documents that follow this convention.

```
<top>

<num> Number: 426
<title> law enforcement, dogs

<desc> Description:
Provide information on the use of dogs worldwide for
law enforcement purposes.

<narr> Narrative:
Relevant items include specific information on the
use of dogs during an operation. Training of dogs
and their handlers are also relevant.

</top>
```

Figure 1.8 TREC topic 426.

Document sets for newer TREC adhoc tasks are often taken from the Web. Until 2009, the largest of these was the 426GB GOV2 collection, which contains 25 million Web pages crawled from sites in the U.S. government's `gov` domain in early 2004. This crawl attempted to reach as many pages as possible within the `gov` domain, and it may be viewed as a reasonable snapshot of that domain within that time period. GOV2 contains documents in a wide variety of formats and lengths, ranging from lengthy technical reports in PDF to pages of nothing but links in HTML. GOV2 formed the document set for the Terabyte Track from TREC 2004 until the track was discontinued at the end of 2006. It also formed the collection for the Million Query Track at TREC 2007 and 2008.

Although the GOV2 collection is substantially larger than any previous TREC collection, it is still orders of magnitude smaller than the collections managed by commercial Web search engines. TREC 2009 saw the introduction of a billion-page Web collection, known as the ClueWeb09 collection, providing an opportunity for IR researchers to work on a scale comparable to commercial Web search.[3]

For each year that a track operates an adhoc task, NIST typically creates 50 new topics. Participants are required to freeze development of their systems before downloading these topics. After downloading the topics, participants create queries from them, run these queries against the document set, and return ranked lists to NIST for evaluation.

A typical TREC adhoc topic, created for TREC 1999, is shown in Figure 1.8. Like most TREC topics, it is structured into three parts, describing the underlying information need in several forms. The *title* field is designed to be treated as a keyword query, similar to a query that might be entered into a search engine. The *description* field provides a longer statement of the topic requirements, in the form of a complete sentence or question. It, too, may be used

[3] `boston.lti.cs.cmu.edu/Data/clueweb09`

Table 1.2 Summary of the test collections used for many of the experiments described in this book.

Document Set	Number of Docs	Size (GB)	Year	Topics
TREC45	0.5 million	2	1998	351–400
			1999	401–450
GOV2	25.2 million	426	2004	701–750
			2005	751–800

as a query, particularly by research systems that apply natural language processing techniques as part of retrieval. The *narrative*, which may be a full paragraph in length, supplements the other two fields and provides additional information required to specify the nature of a relevant document. The narrative field is primarily used by human assessors, to help determine if a retrieved document is relevant or not.

Most retrieval experiments in this book report results over four TREC test collections based on two document sets, a small one and a larger one. The small collection consists of the documents from disks 4 and 5 of the TREC CDs described above, excluding the documents from the *Congressional Record*. It includes documents from the *Financial Times*, the U.S. *Federal Register*, the U.S. Foreign Broadcast Information Service, and the *LA Times*. This document set, which we refer to as *TREC45*, was used for the main adhoc task at TREC 1998 and 1999.

In both 1998 and 1999, NIST created 50 topics with associated relevance judgments over this document set. The 1998 topics are numbered 350–400; the 1999 topics are numbered 401–450. Thus, we have two test collections over the TREC45 document set, which we refer to as *TREC45 1998* and *TREC45 1999*. Although there are minor differences between our experimental procedure and that used in the corresponding TREC experiments (which we will ignore), our experimental results reported over these collections may reasonably be compared with the published results at TREC 1998 and 1999.

The larger one of the two document sets used in our experiments is the GOV2 corpus mentioned previously. We take this set together with topics and judgments from the TREC Terabyte track in 2004 (topics 701–750) and 2005 (751–800) to form the GOV2 2004 and GOV2 2005 collections. Experimental results reported over these collections may reasonably be compared with the published results for the Terabyte track of TREC 2004 and 2005.

Table 1.2 summarizes our four test collections. The TREC45 collection may be obtained from the NIST Standard Reference Data Products Web page as Special Databases 22 and 23.[4] The GOV2 collection is distributed by the University of Glasgow.[5] Topics and qrels for these collections may be obtained from the TREC data archive.[6]

[4] `www.nist.gov/srd`

[5] `ir.dcs.gla.ac.uk/test_collections`

[6] `trec.nist.gov`

1.5 Open-Source IR Systems

There exists a wide variety of open-source information retrieval systems that you may use for exercises in this book and to start conducting your own information retrieval experiments. As always, a (non-exhaustive) list of open-source IR systems can be found in Wikipedia.[7]

Since this list of available systems is so long, we do not even try to cover it in detail. Instead, we restrict ourselves to a very brief overview of three particular systems that were chosen because of their popularity, their influence on IR research, or their intimate relationship with the contents of this book. All three systems are available for download from the Web and may be used free of charge, according to their respective licenses.

1.5.1 Lucene

Lucene is an indexing and search system implemented in Java, with ports to other programming languages. The project was started by Doug Cutting in 1997. Since then, it has grown from a single-developer effort to a global project involving hundreds of developers in various countries. It is currently hosted by the Apache Foundation.[8] Lucene is by far the most successful open-source search engine. Its largest installation is quite likely Wikipedia: All queries entered into Wikipedia's search form are handled by Lucene. A list of other projects relying on its indexing and search capabilities can be found on Lucene's "PoweredBy" page.[9]

Known for its modularity and extensibility, Lucene allows developers to define their own indexing and retrieval rules and formulae. Under the hood, Lucene's retrieval framework is based on the concept of *fields*: Every document is a collection of fields, such as its title, body, URL, and so forth. This makes it easy to specify structured search requests and to give different weights to different parts of a document.

Due to its great popularity, there is a wide variety of books and tutorials that help you get started with Lucene quickly. Try the query "lucene tutorial" in your favorite Web search engine.

1.5.2 Indri

Indri[10] is an academic information retrieval system written in C++. It is developed by researchers at the University of Massachusetts and is part of the Lemur project,[11] a joint effort of the University of Massachusetts and Carnegie Mellon University.

[7] `en.wikipedia.org/wiki/List_of_search_engines`
[8] `lucene.apache.org`
[9] `wiki.apache.org/lucene-java/PoweredBy`
[10] `www.lemurproject.org/indri/`
[11] `www.lemurproject.org`

Indri is well known for its high retrieval effectiveness and is frequently found among the top-scoring search engines at TREC. Its retrieval model is a combination of the language modeling approaches discussed in Chapter 9. Like Lucene, Indri can handle multiple fields per document, such as title, body, and anchor text, which is important in the context of Web search (Chapter 15). It supports automatic query expansion by means of pseudo-relevance feedback, a technique that adds related terms to an initial search query, based on the contents of an initial set of search results (see Section 8.6). It also supports query-independent document scoring that may, for instance, be used to prefer more recent documents over less recent ones when ranking the search results (see Sections 9.1 and 15.3).

1.5.3 Wumpus

Wumpus[12] is an academic search engine written in C++ and developed at the University of Waterloo. Unlike most other search engines, Wumpus has no built-in notion of "documents" and does not know about the beginning and the end of each document when it builds the index. Instead, every part of the text collection may represent a potential unit for retrieval, depending on the structural search constraints specified in the query. This makes the system particularly attractive for search tasks in which the ideal search result may not always be a whole document, but may be a section, a paragraph, or a sequence of paragraphs within a document.

Wumpus supports a variety of different retrieval methods, including the proximity ranking function from Chapter 2, the BM25 algorithm from Chapter 8, and the language modeling and divergence from randomness approaches discussed in Chapter 9. In addition, it is able to carry out real-time index updates (i.e., adding/removing files to/from the index) and provides support for multi-user security restrictions that are useful if the system has more than one user, and each user is allowed to search only parts of the index.

Unless explicitly stated otherwise, all performance figures presented in this book were obtained using Wumpus.

1.6 Further Reading

This is not the first book on the topic of information retrieval. Of the older books providing a general introduction to IR, several should be mentioned. The classic books by Salton (1968) and van Rijsbergen (1979) continue to provide insights into the foundations of the field. The treatment of core topics given by Grossman and Frieder (2004) remains relevant. Witten et al. (1999) provide background information on many related topics that we do not cover in this book, including text and image compression.

[12] `www.wumpus-search.org`

Several good introductory texts have appeared in recent years. The textbook by Croft et al. (2010) is intended to give an undergraduate-level introduction to the area. Baeza-Yates and Ribeiro-Neto (2010) provide a broad survey of the field, with experts contributing individual chapters on their areas of expertise. Manning et al. (2008) provide another readable survey.

Survey articles on specific topics appear regularly as part of the journal series *Foundations and Trends in Information Retrieval.* The *Encyclopedia of Database Systems* (Özsu and Liu, 2009) contains many introductory articles on topics related to information retrieval. Hearst (2009) provides an introduction to user interface design for information retrieval applications. The field of natural language processing, particularly the sub-field of statistical natural language processing, is closely related to the field of information retrieval. Manning and Schütze (1999) provide a thorough introduction to that area.

The premier research conference in the area is the *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR), now well into its fourth decade. Other leading research conferences and workshops include the *ACM Conference on Information and Knowledge Management* (CIKM), the *Joint Conference on Digital Libraries* (JCDL), the *European Conference on Information Retrieval* (ECIR), the *ACM International Conference on Web Search and Data Mining* (WSDM), the conference on *String Processing and Information Retrieval* (SPIRE), and the *Text REtrieval Conference* (TREC). Important IR research also regularly appears in the premier venues of related fields, such as the *World Wide Web Conference* (WWW), the *Annual Conference on Neural Information Processing Systems* (NIPS), the *Conference on Artificial Intelligence* (AAAI), and the *Knowledge Discovery and Data Mining Conference* (KDD). The premier IR journal is *ACM Transactions on Information Systems.* Other leading journals include *Information Retrieval* and the venerable *Information Processing & Management.*

Many books and Web sites devoted to learning and using XML are available. One important resource is the XML home page of the World Wide Web Consortium (W3C),[13] the organization responsible for defining and maintaining the XML technical specification. Along with extensive reference materials, the site includes pointers to introductory tutorials and guides. Jon Bosak's personal Web page[14] contains many articles and other information related to the early development and usage of XML, including Shakespeare's plays.

[13] `www.w3.org/XML/`

[14] `www.ibiblio.org/bosak`

1.7 Exercises

Exercise 1.1 Record the next ten queries you issue to a Web search engine (or check your Web history if your search engine allows it). Note how many are *refinements* of previous queries, adding or removing terms to narrow or broaden the focus of the query. Choose three commercial search engines, including the engine you normally use, and reissue the queries on all three. For each query, examine the top five results from each engine. For each result, rate on it a scale from -10 to +10, where +10 represents a perfect or ideal result and -10 represents a misleading or harmful result (spam). Compute an average score for each engine over all ten queries and results. Did the engine you normally use receive the highest score? Do you believe that the results of this exercise accurately reflect the relative quality of the search engines? Suggest three possible improvements to this experiment.

Exercise 1.2 Obtain and install an open-source IR system, such as one of those listed in Section 1.5. Create a small document collection from your e-mail, or from another source. A few dozen documents should be enough. Index your collection. Try a few queries.

Exercise 1.3 Starting in state 3 of the Markov model in Figure 1.6, what is the probability of generating "not not be to be"?

Exercise 1.4 Starting in an unknown state, the Markov model in Figure 1.6 generates "to be". What state or states could be the current state of the model after generating this text?

Exercise 1.5 Is there a finite n, such that the current state of the Markov model in Figure 1.6 will always be known after it generates a string of length n or greater, regardless of the starting state?

Exercise 1.6 For a given Markov model, assume there is a finite n so that the current state of the model will always be known after it generates a string of length n or greater. Describe a procedure for converting such a Markov model into an nth-order finite-context model.

Exercise 1.7 Assume that we extend Markov models to allow multiple transitions between pairs of states, where each transition between a given pair of states is labeled with a different term. For any such model, show that there is an equivalent Markov model in which there is no more than one transition between any pair of states. (*Hint*: Split target states.)

Exercise 1.8 Outline a procedure for converting an nth-order finite-context language model into a Markov model. How many states might be required?

Exercise 1.9 (project exercise) This exercise develops a test corpus, based on Wikipedia, that will be used in a number of exercises throughout Part I.

To start, download a current copy of the English-language Wikipedia. At the time of writing, its downloadable version consisted of a single large file. Wikipedia itself contains documentation describing the format of this download.[15]

Along with the text of the articles themselves, the download includes *redirection records* that provide alternative titles for articles. Pre-process the download to remove these records and any other extraneous information, leaving a set of individual articles. Wikipedia-style formatting should also be removed, or replaced with XML-style tags of your choosing. Assign a unique identifier to each article. Add <DOC> and <DOCNO> tags. The result should be consistent with TREC conventions, as described in Section 1.4 and illustrated in Figure 1.7.

Exercise 1.10 (project exercise) Following the style of Figure 1.8, create three to four topics suitable for testing retrieval performance over the English-language Wikipedia. Try to avoid topics expressing an information need that can be completely satisfied by a single "best" article. Instead, try to create topics that require multiple articles in order to cover all relevant information. (*Note*: This exercise is suitable as the foundation for a class project to create a Wikipedia test collection, with each student contributing enough topics to make a combined set of 50 topics or more. (See Exercise 2.13 for further details.)

Exercise 1.11 (project exercise) Obtain and install an open-source IR system (see Exercise 1.2). Using this IR system, index the collection you created in Exercise 1.9. Submit the titles of the topics you created in Exercise 1.10 as queries to the system. For each topic, judge the top five documents returned as either relevant or not. Does the IR system work equally well on all topics?

Exercise 1.12 (project exercise) Tokenize the collection you created in Exercise 1.9, following the procedure of Section 1.3.2. For this exercise, discard the tags and keep only the words (i.e., strings of alphanumeric characters). Wikipedia text is encoded in UTF-8 Unicode, but you may treat it as ASCII text for the purpose of this exercise. Generate a log-log plot of frequency vs. rank order, equivalent to that shown in Figure 1.5. Does the data follow Zipf's law? If so, what is an approximate value for α?

Exercise 1.13 (project exercise) Create a trigram language model based on the tokenization of Exercise 1.12. Using your language model, implement a random Wikipedia text generator. How could you extend your text generator to generate capitalization and punctuation, making your text look more like English? How could you extend your text generator to generate random Wikipedia articles, including tagging and links?

[15] `en.wikipedia.org/wiki/Wikipedia:Database_download`

1.8 Bibliography

Baeza-Yates, R. A., and Ribeiro-Neto, B. (2010). *Modern Information Retrieval* (2nd ed.). Reading, Massachusetts: Addison-Wesley.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search Engines: Information Retrieval in Practice*. London, England: Pearson.

Grossman, D. A., and Frieder, O. (2004). *Information Retrieval: Algorithms and Heuristics* (2nd ed.). Berlin, Germany: Springer.

Hearst, M. A. (2009). *Search User Interfaces*. Cambridge, England: Cambridge University Press.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press.

Manning, C. D., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press.

Özsu, M. T., and Liu, L., editors (2009). *Encyclopedia of Database Systems*. Berlin, Germany: Springer.

Salton, G. (1968). *Automatic Information Organziation and Retrieval*. New York: McGraw-Hill.

van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). London, England: Butterworths.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd ed.). San Francisco, California: Morgan Kaufmann.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least-Effort*. Cambridge, Massachusetts: Addison-Wesley.